Probabilistic Business Constraints and their Monitoring

Fabrizio Maria Maggi[1], Marco Montali[2], Rafael Peñaloza[3], and Anti Alman[1]

[1] University of Tartu, Estonia
{f.m.maggi,anti.alman}@ut.ee
[2] Free University of Bozen-Bolzano, Italy
montali@inf.unibz.it
[3] University of Milano-Bicocca
rafael.penaloza@unimib.it

Abstract. Monitoring and conformance checking are fundamental tasks to detect deviations between the actual and the expected courses of execution in a business process. In a variety of application domains, the model capturing the expected behaviors may be intrinsically uncertain, e.g., to distinguish between standard courses of execution and exceptional but still conforming ones. Surprisingly, only very few approaches consider uncertainty as a first-class citizen in this spectrum. In this paper, we tackle this timely and challenging problem considering the setting where the model is described as a set of declarative, temporal business constraints. First, we delve into the conceptual meaning of probabilistic business constraints, and argue that they can be discovered from event data using already existing techniques. Second, we study how to monitor probabilistic constraints, where constraints and their combinations may be in multiple monitoring states at the same time, though with different associated probabilities. Third, we operationalize this monitoring framework using automata coupled with probabilities, and report on its actual implementation.

Keywords: Business Process Monitoring · Probabilistic Declarative Process Models · Probabilistic Conformance Checking

1 Introduction

A key functionality that any process-aware information system should support is *compliance monitoring* [9], i.e., the ability to verify at runtime whether the actual flow of work is compliant with the intended business process model. This runtime form of conformance checking is particularly suited to recognize and handle deviations on still running instances. A common way of representing monitoring requirements that capture the expected behavior of a process, is by using declarative, business constraints. A plethora of studies has demonstrated that, in several settings, business constraints can be formalized in terms of temporal logic rules. Within this paradigm, the *Declare* constraint-based process modeling language [18] has been introduced as a front-end language to specify business constraints based on Linear Temporal Logic over finite traces (LTL_f) [2]. The advantage of this approach is that the corresponding automata-theoretic characterization of LTL_f can be exploited to provide advanced monitoring facilities to continuously determine the state of constraints based on the events collected

at runtime [13,1], and to detect violations at the earliest moment possible by identifying conflicting constraints [14].

In a variety of application domains, business constraints naturally come with uncertainty. Consider, e.g., constraints: *(i)* expressing best practices that have to be followed in most but not necessarily all the cases; *(ii)* partially reflecting the behavior of uncontrollable, external stakeholders; *(iii)* capturing exceptional but still conforming courses of execution. Constraints are also inherently uncertain when they are discovered from event data. Surprisingly, only very few approaches consider uncertainty as a first-class citizen not only when it comes to monitoring and conformance checking of business constraints, but also when more conventional procedural notations are considered [8].

In the context of business constraints expressed with temporal logics, introducing uncertainty is notoriously known to be extremely challenging. Combined logics in this spectrum thus come with semantic or syntactic restrictions to tame the interaction of temporal operators and probabilities (see, e.g., [17,6]). In addition, such logics do not lend themselves to be applied in the context of business processes, since they are defined over infinite traces. To tackle these issues, the probabilistic temporal logic over finite traces PLTL$_f$, and its fragment PLTL$_f^0$, have been recently proposed in [12].[4] PLTL$_f^0$ is of particular interest here, since the way probabilities are used can be interpreted by naturally matching the notion of conformance: a constraint φ has probability p, if, by imagining all the traces contained in a log, the fraction p of such traces satisfies φ.

Leveraging on this, we provide here a threefold contribution. First, we delve into the conceptual meaning of probabilistic business constraints, introducing probabilities in Declare, obtaining the ProbDeclare language. We also argue that probabilistic Declare constraints can be discovered from event data using, off-the-shelf, already existing techniques, with strong guarantees on the consistency of the generated models. Interestingly, this is done by interpreting the event log as a stochastic language, in the style of [8]. Second, we study how to monitor probabilistic constraints, where constraints and their combinations may be in multiple monitoring states at the same time, though with different associated probabilities. This is based on the fact that a single ProbDeclare model gives raise to multiple scenarios, each with its own distinct probability, where some of the constraints are expected to be satisfied, and the others to be violated. Third, we operationalise this monitoring framework using automata coupled with probabilities, and report on its actual implementation.

The paper is structured as follows. In Section 2, we recall the temporal logics used in the remainder of the paper. In Section 3, we introduce the ProbDeclare approach and study the notion of constraints scenarios and their probabilities. In Section 4, we discuss how ProbDeclare constraints can be seamlessly discovered from event data using existing techniques. In Section 5, we turn ProbDeclare into a monitoring framework, which has been fully implemented.

2 A Gentle Introduction to the PLTL$_f^0$ Logic

Since PLTL$_f$ builds on LTL$_f$, we start by briefly recalling LTL$_f$.

LTL over finite traces. LTL$_f$ has exactly the same syntax of standard LTL, but, differently from LTL, it interprets formulae over an unbounded, yet finite linear sequence

[4] The paper is available for reading at [11].

of states. Given an alphabet Σ of atomic propositions (in our setting, representing activities), an LTL$_f$ formula φ is built by extending propositional logic with temporal operators:

$$\varphi ::= a \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \,\mathcal{U}\, \varphi_2 \quad \text{where } a \in \Sigma.$$

The semantics of LTL$_f$ is given in terms of *finite traces* denoting finite, *possibly empty*, sequences $\tau = \tau_0, \ldots, \tau_n$ of elements from the alphabet 2^Σ, containing all possible propositional interpretations of the propositional symbols in Σ. In the context of this paper, consistently with the literature on business process execution traces, we make the simplifying assumption that in each point of the sequence, one and only one element from Σ holds. Under this assumption, τ becomes a total sequence of activity occurrences from Σ, matching the standard notion of (process) execution trace. The evaluation of a formula is done in a given state (i.e., position) of the trace, and we use the notation $\tau, i \models \varphi$ to express that φ holds in the position i of τ.

In the syntax above, operator $\bigcirc$ denotes the *next state* operator, and $\bigcirc\varphi$ is true, if there exists a next state (i.e., the current state is not at the end of the trace), and in the next state φ holds. Operator $\mathcal{U}$ instead is the *until* operator, and $\varphi_1 \,\mathcal{U}\, \varphi_2$ is true, if φ_1 holds now and continues to hold until eventually, in a future state, φ_2 holds. From the given syntax, we can derive the usual boolean operators $\wedge$ and $\rightarrow$, the two formulae $true$ and $false$, as well as additional temporal operators. We consider, in particular, the following three:

- (eventually) $\Diamond\varphi = true \,\mathcal{U}\, \varphi$ is true, if there is a future state where φ holds;
- (globally) $\Box\varphi = \neg\Diamond\neg\varphi$ is true, if now and in all future states φ holds;
- (weak until) $\varphi_1 \,\mathcal{W}\, \varphi_2 = \varphi_1 \,\mathcal{U}\, \varphi_2 \vee \Box\varphi_1$ relaxes the until operator by admitting the possibility that φ_2 never becomes true, in this case by requiring that φ_1 holds now and in all future states.

We write $\tau \models \varphi$ as a shortcut notation for $\tau, 0 \models \varphi$, and say that formula φ is *satisfiable*, if there exists a trace τ such that $\tau \models \varphi$.

Example 1. The LTL$_f$ formula $\Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{accept})$ models that, whenever an order is closed, then it is eventually accepted. The structure of the formula follows what is called *response template* in Declare. ◁

Every LTL$_f$ formula φ can be translated into a corresponding standard finite-state automaton $\mathcal{A}_\varphi$ that accepts all and only those finite traces that satisfy φ [2,1]. Although the complexity of reasoning with LTL$_f$ is the same as that of LTL, finite-state automata are much easier to manipulate in comparison with Büchi automata, which are necessary when formulae are interpreted over infinite traces. This is the main reason why LTL$_f$ has been extensively and successfully adopted within BPM to capture constraint-based, declarative processes, in particular providing the formal basis of the *Declare* modeling language [18]. Specifically, automata-based techniques for LTL$_f$ have been adopted to tackle fundamental tasks within the lifecycle of Declare processes, such as consistency checking [18,16], enactment and monitoring [18,13,1], and discovery support [10].

Probabilistic LTL over finite traces, and its fragment PLTL$_f^0$. The PLTL$_f$ logics [12] extends LTL$_f$ with a special *probabilistic next* operator, which intuitively indicates that in the next time point a formula holds with a given probability (or range of probabilities). This is interpreted in a so-called *superposition* semantics, where every trace is possible (with different probabilities) until actual events are observed. The main drawback of the whole logic is that the interaction of probabilities and time causes

a complexity jump from PSPACE to EXPTIME for all standard reasoning tasks, making it impossible to adopt the well-established automata-based techniques used in LTL_f. To tackle this issue, [12] also brings forward a fragment of this logic, called PLTL_f^0. This fragment *is particularly suitable to capture business constraints and interpret probabilities statistically, as the fraction of conforming vs non-conforming traces*.

Specifically, a PLTL_f^0 formula is a set of LTL_f formulae, each one coming with a probability on the likelihood of its satisfaction.

Definition 1. *A PLTL_f^0 formula is a set of logical expressions of the form $\circledcirc_p\varphi$, where φ is a classical LTL_f formula, $\bowtie \in \{=, \neq, \leq, \geq, <, >\}$, and $p \in [0, 1]$.* ◁

The special $\circledcirc$ operator indicates that φ holds with probability $\bowtie\ p$, giving to p the interpretation of a *statistical probability*. In fact, we can read this expression as the fact that the *fraction of execution traces satisfying constraint φ is $\bowtie\ p$*. Only for simplicity of presentation, in the remainder of this paper, we restrict our attention to expressions where we have exact probabilities, i.e., where $\bowtie$ is the equality operator. For compactness, we write $\circledcirc_p\varphi$ as a shortcut notation for $\circledcirc_{=p}\varphi$. We stress here once and for all that thanks to the logical machinery introduced in [12], all the presented results in this paper carry over general PLTL_f^0 expressions employing arbitrary comparison operators.

This statistical interpretation of probabilities is central in the context of the present paper, and leads to the following, fundamental observation, which is key to understand how probabilistic constraints work: the PLTL_f^0 formula $\circledcirc_p\varphi$ is logically equivalent to the corresponding inverse formula $\circledcirc_{1-p}\neg\varphi$. This reflects the intuition that, whenever φ holds in a fraction p of traces from an event log, then $\neg\varphi$ must hold in the complementary fraction $1-p$ of traces from that log. Conversely, given an unknown execution trace τ, trace τ will satisfy φ with probability p, and will violate φ (i.e., satisfy $\neg\varphi$) with probability $1-p$. Hence, we can interpret φ and $\neg\varphi$ as two alternative, *possible worlds*, each coming with its own probability (respectively, p and $1-p$).

Since in PLTL_f^0 time and probability mix together, satisfiability depends on both.

Definition 2. *A PLTL_f^0 expression $\circledcirc_p\varphi$ is satisfiable, if $p \neq 0$ and φ is satisfiable in the classical LTL_f sense.* ◁

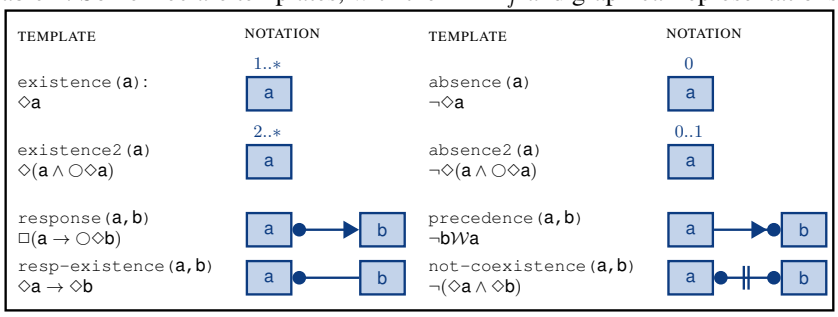
As a consequence of this definition, we get that a PLTL_f^0 expression $\circledcirc_1\varphi$ with probability 1 corresponds to just one possible world, where φ is true.

The notion of possible world introduced above carries over general PLTL_f^0 formulae with multiple expressions. This is discussed in Section 3.

Example 2. The PLTL_f^0 formula $\circledcirc_{0.8}\Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{accept})$ models that the response formula $\Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{accept})$ has probability 0.8, that is, in 80% of the process traces, it is true that, whenever an order is closed, then it is eventually accepted. This is equivalent to formula $\circledcirc_{0.2}\Diamond(\mathsf{close} \wedge \neg\bigcirc\Diamond\mathsf{accept}$, which asserts that in 20% of the traces the response is violated, i.e., there exists a state where the order is closed and not accepted afterwards. Given a trace τ, there is then 0.8 chance that τ will satisfy the response formula $\Box(\mathsf{close} \rightarrow \bigcirc\Diamond\mathsf{accept})$, and 0.2 that τ will violate such a formula (i.e., satisfy its negation $\Diamond(\mathsf{close} \wedge \neg\bigcirc\Diamond\mathsf{accept})$). ◁

PLTL_f^0 is interesting in the context of this paper for a twofold reason. On the one hand, reasoning about PLTL_f^0 falls back to PSPACE, matching the complexity of the classical LTL_f case (without probabilities). As we will show in this paper, this allows us to resort to standard LTL_f automata-based techniques for probabilistic monitoring

Table 1: Some Declare templates, with their LTL_f and graphical representations.



and conformance checking. On the other hand, thanks to its semantics, PLTL_f^0 can be used to seamlessly obtain a probabilistic variant of Declare. This is tackled next.

3 Probabilistic Declarative Process Modeling

We employ PLTL_f^0 as the basis for obtaining a probabilistic version of Declare [18,15], and for reasoning on probabilistic constraints.

3.1 The ProbDeclare Framework

Declare is a constraint-based process modeling language based on LTL_f. Differently from imperative process modeling languages, Declare models a process by fixing a set of activities, and defining a set of *temporal constraints* over them, accepting every execution trace that satisfies all constraints. Constraints are specified via pre-defined LTL_f templates, which come with a corresponding graphical representation (see Table 1 for the Declare patterns we use in this paper). For the sake of generality, in the remainder of the paper, we consider arbitrary LTL_f formulae as constraints, but, in the examples, we work on formulae whose templates can be represented graphically in Declare.

We lift Declare to its probabilistic version ProbDeclare as follows.

Definition 3. *A ProbDeclare model is a pair $\langle \Sigma, \mathcal{C} \rangle$, where Σ is a set of activities and $\mathcal{C}$ is a set of probabilistic constraints. Each* probabilistic constraint *in $\mathcal{C}$ is a pair $\langle \varphi, p \rangle$, where φ is an LTL_f formula over Σ representing the* constraint formula*, and p is a rational value in $[0,1]$ representing the* constraint probability. ◁

Formally, a ProbDeclare model $\langle \Sigma, \{\langle \varphi_1, p_1 \rangle, \ldots, \langle \varphi_n, p_n \rangle\} \rangle$ is simply the PLTL_f^0 formula $\{ \circledcirc_{p_1} \varphi_1, \ldots, \circledcirc_{p_n} \varphi_n \}$, where each expression corresponds to a probabilistic constraint. By recalling that, in PLTL_f^0, expressions $\circledcirc_b \varphi$ and $\circledcirc_{1-p} \neg \varphi$ are equivalent, we have that, in ProbDeclare, constraints $\langle \varphi, p \rangle$ and $\langle \neg \varphi, 1-p \rangle$ are equivalent. This has the effect that, in ProbDeclare, the distinction between `existence` and `absence` templates (cf. the first two lines of Table 1) gets blurred. In fact:

$$\langle \texttt{existence(a)}, p \rangle = \langle \Diamond a, p \rangle = \langle \neg \Diamond a, 1-p \rangle = \langle \texttt{absence}(a), 1-p \rangle$$

The same line of reasoning can be applied for the other comparison operators, and considering the `existence2` and `absence2` templates. Such constraints have in fact to be interpreted in the light of *probability of (repeated) presence* for a given activity.

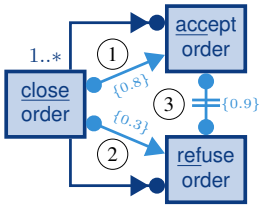
Example 3. A small but illustrative ProbDeclare model is shown in the top left of Figure 1. Crisp constraints with probability 1 are shown in dark blue, and genuine probabilistic constraints are shown in light blue, with probability values attached. In the model, we express that each order is at some point closed, and, whenever this happens, there is probability 0.8 that it will be eventually accepted, and probability 0.3 that it will be eventually refused. Notice that the sum of these probabilities exceed 1, so even if we have still to define formally how constraints and their probabilities interplay with each other, we can already see that, in a small fraction of traces, there will be an acceptance and also a rejection (capturing the fact that a previous decision on a closed order was subverted later on). On the other hand, there is a sensible amount of traces where the order will be eventually accepted, but not refused, given the fact that the `response` constraint connecting close order to refuse order is only of 0.3. In 90% of the cases, it is asserted that acceptance and rejection are mutually exclusive. Finally, accepting or rejecting an order can only occur if the order has been already closed. ◁

3.2 Constraints Scenarios and their Probabilities

Since a ProbDeclare model contains multiple probabilistic constraints, we have to consider that, probabilistically, a trace may satisfy or violate each of the constraints contained in the model, thus yielding multiple possible worlds, each one defining which constraints are satisfied, and which violated. E.g., in Figure 1 we may have a trace containing close order followed by accept order and refuse order, thus violating the `not-coexistence` constraint relating acceptance and refusal. This is indeed possible in 10% of the traces. More in general, consider a ProbDeclare model $M = \langle \Sigma, \{\langle \varphi_1, p_1 \rangle, \ldots, \langle \varphi_n, p_n \rangle\}\rangle$. Each constraint formula φ_i is satisfied by a trace with probability p_i, and violated with probability $1 - p_i$. Hence, a model of this form implicitly yields, potentially, 2^n possible worlds resulting from all possible choices of which constraints formulae are satisfied, and which are violated (recall that violating a formula means satisfying its negation). We call such possible worlds *constraint scenarios*. The key point is to understand which scenarios are plausible, and with which probability, considering the probabilities attached to the various constraints.

If a constraint has probability 1, we do not need to consider the two alternatives, since every trace will need to satisfy its formula. Hence, to identify a scenario, we proceed as follows. We consider the $m \leq n$ constrains with probability different than 1, and fix an order over them. Then, a scenario is defined by a number between 0 and $m - 1$, whose corresponding binary representation defines which constraint formulae are satisfied, and which violated: specifically, for constraint formula φ_i of index i, if the bit in position $i - 1$ is 1, then the scenario contains φ_i, if instead that bit is 0, then the scenario contains $\neg\varphi_i$. The overall formula describing a scenario is then simply the conjunction of all such formulae, together with all the formulae of constraints with probability 1. Clearly, each execution trace belongs to one and only one constraint scenario: it does so, when it satisfies the conjunctive formula associated to that scenario. We then say that a scenario is *logically plausible*, if such a conjunctive formula is satisfiable: if it is not, then the scenario has to be discarded, since no trace will ever belong to it.

Example 4. Figure 1 shows a ProbDeclare model with 6 constraints, three of which are crisp constraints with probability 1, while the other three are genuinely proba-



	①	②	③	SAT?
S_{000}	◇(close ∧ ¬○◇acc)	◇(close ∧ ¬○◇ref)	◇acc ∧ ◇refuse	no
S_{001}	◇(close ∧ ¬○◇acc)	◇(close ∧ ¬○◇ref)	¬(◇acc ∧ ◇refuse)	**yes**
S_{010}	◇(close ∧ ¬○◇acc)	□(close → ○◇ref)	◇acc ∧ ◇refuse	no
S_{011}	◇(close ∧ ¬○◇acc)	□(close → ○◇ref)	¬(◇acc ∧ ◇refuse)	**yes**
S_{100}	□(close → ○◇acc)	◇(close ∧ ¬○◇ref)	◇acc ∧ ◇refuse	no
S_{101}	□(close → ○◇acc)	◇(close ∧ ¬○◇ref)	¬(◇acc ∧ ◇refuse)	**yes**
S_{110}	□(close → ○◇acc)	□(close → ○◇ref)	◇acc ∧ ◇refuse	**yes**
S_{111}	□(close → ○◇acc)	□(close → ○◇ref)	¬(◇acc ∧ ◇refuse)	no

Fig. 1: A ProbDeclare model, with 8 constraint scenarios, out of which only 4 of which are logically plausible. Recall that each scenario implicitly contains also the three constraint formulae derived from the three constraints with probability 1.

bilistic constraints. The circled numbers represent the ordering of such constraints. 8 possible constraint scenarios have to be considered, each one enforcing the satisfaction of the three crisp constraints, while picking the satisfaction or violation of the three probabilistic constraints `response`(close, acc), `response`(close, ref), and `not-coexistence`(acc, ref). Logically speaking, recall that violating a constraint means satisfying its negation, and so we have to consider 6 different formulae: `response`(close, acc) $= \Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})$, and its negation is $\Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{acc})$ (similarly for `response`(close,ref)); `not-coexistence`(acc, ref) $= \neg(\Diamond\text{acc} \wedge \Diamond\text{refuse})$, and its negation is $\Diamond\text{acc} \wedge \Diamond\text{refuse}$.

The resulting scenarios are reported in the same figure, using the naming conventions introduced before. For example, scenario S_{111} is the one where all constraints are satisfied, whereas scenario S_{101} is the scenario that satisfies `response`(close, acc) and `not-coexistence`(acc, ref), but violates `response`(close, ref).

By checking the logical satisfiability of the conjunction of the formulae entailed by a given scenario, we can see whether the scenario is logically plausible. As shown in Figure 1, only 4 scenarios are actually logically plausible. For example, S_{111} is *not* logically plausible. In fact, it requires that the order is closed (due to the crisp $1..*$ constraint on close order) and, consequently, that the order is eventually accepted and refused (due to the two response constraints attached to close order, which in this scenario must be both satisfied); however, the presence of both an acceptance and a refusal violates the `not-coexistence` constraint linking such two activities, contradicting the requirement that also this constraint must be satisfied in this scenario. S_{101} is logically plausible: it is satisfied by the trace where an order is closed and then accepted.

All in all, we have 4 logically plausible scenarios: *(i)* S_{001}, where an order is closed and later not accepted nor refused; *(ii)* S_{011}, where an order is closed and later refused (and not accepted); *(iii)* S_{101}, where an order is closed and later accepted (and not refused); *(iv)* S_{110}, where an order is closed and later accepted and refused. ◁

While it is clear that a logically implausible scenario should correspond to probability 0, are all logically plausible scenarios really plausible when the actual constraint probabilities are taken into account? By looking at Figure 1, one can notice that scenario S_{001} is logically plausible: it describes traces where an order is closed but not accepted nor refused. As we will see, however, this cannot happen given the actual probabilities of 0.8 and 0.3 attached to `response`(close, acc) and `response`(close, ref).

More in general, what is the probability of a constraint scenario, i.e., the fraction of traces that belong to that scenario? What are the probability values that actually admit

the existence of scenarios, and those that instead lead to an inconsistent ProbDeclare model? To answer these questions, we resort to the probability calculation technique in [12]. We associate each scenario to a probability variable, using the same naming convention (e.g., scenario S_{001} mentioned before corresponds to variable x_{001}). Given a ProbDeclare model $M = \langle \Sigma, \{\langle \varphi_1, p_1 \rangle, \ldots, \langle \varphi_n, p_n \rangle\}\rangle$, we then construct the system $\mathcal{L}_M$ of inequalities as follows:

$$\begin{aligned} x_i &\geq 0 && 0 \leq i < 2^n \\ \textstyle\sum_{i=0}^{2^n-1} x_i &= 1 && \\ \textstyle\sum_{j\text{th position is 1}} x_i &= p_j && 0 \leq j < n \\ x_i &= 0 && \text{if scenario } S_i \text{ is logically implausible} \end{aligned}$$

The first two lines guarantee that we assign a non-negative value to each variable, and that their sum is one. We can see these assignments as probabilities, having the guarantee that all scenarios together cover the full spectrum. The third line verifies the probability associated to each constraint in M. In particular, it constructs one equation per constraint $\langle \varphi_j, p_j \rangle$ in M, ensuring that all the variables that correspond to scenarios making φ_i true should all together yield its probability p_i. The last line ensures that logically implausible scenarios get assigned probability 0.

First and foremost, we can use this system of inequalities to check whether a given ProbDeclare model is consistent: M is consistent if and only if $\mathcal{L}_M$ admits a solution. In fact, solving $\mathcal{L}_M$ corresponds to verifying whether the class of all possible traces can be divided in such a way that the proportions required by the probabilistic constraints in the different scenarios are satisfied.

Example 5. Consider the ProbDeclare model M containing two constraints:

1. `existence`(close)=$\Diamond$close with probability 0.1;
2. `response`(close,accept)=$\Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})$ with probability 0.8.

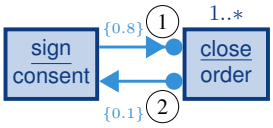
M indicates that only 10% of the traces contain that the order is closed, and that 80% of the traces are so that, whenever an order is closed, it is eventually accepted. This model is inconsistent. We can see this intuitively by reasoning as follows. The fact that in 80% of the traces, whenever an order is closed, it is eventually accepted, is equivalent to say that, in 20% of the traces, we violate such a response constraint, i.e., we have that an order is closed but then not accepted. All such traces satisfy the `existence` constraint over the close order activity, and, consequently, the probability of such a constraint must be at least 0.2. However, this is contradicted by the first constraint of M, which imposes that such a probability is 0.1.

We now show how this is detected formally. M yields 4 constraint scenarios:

$$S_{00} = \{\neg\Diamond\text{close}, \Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{acc})\} \quad S_{01} = \{\neg\Diamond\text{close}, \Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})\}$$
$$S_{10} = \{\Diamond\text{close}, \Diamond(\text{close} \wedge \neg\bigcirc\Diamond\text{acc})\} \quad S_{11} = \{\Diamond\text{close}, \Box(\text{close} \rightarrow \bigcirc\Diamond\text{acc})\}$$

Scenario S_{00} is logically implausible: it requires and forbids that the order is closed; the other scenarios are instead all logically plausible. Hence, the equations of $\mathcal{L}_M$ are:

$$\begin{aligned} x_{00} + x_{01} + x_{10} + x_{11} &= 1 \\ x_{10} + x_{11} &= 0.1 \\ x_{01} + x_{11} &= 0.8 \\ x_{00} &= 0 \end{aligned}$$



	①	②	SAT?
S_{00}	¬sign $\mathcal{U}$ close	◇(close ∧ ¬○◇sign)	**yes**
S_{01}	¬sign $\mathcal{U}$ close	□(close → ○◇sign)	**yes**
S_{10}	¬close $\mathcal{W}$ sign	◇(close ∧ ¬○◇sign)	**yes**
S_{11}	¬close $\mathcal{W}$ sign	□(close → ○◇sign)	**yes**

Fig. 2: A ProbDeclare model and its 4 constraint scenarios.

The equations yield $x_{10} = 0.2$, $x_{01} = 0.9$, and $x_{11} = -0.1$. This is an inconsistent probability assignment, and witnesses that it is not possible to properly assign suitable fractions of traces to the various constraint scenarios. ◁

When $\mathcal{L}_M$ is solvable, M is consistent. In addition, the solutions of $\mathcal{L}_M$ tell us what is the probability (or range of probabilities) for each constraint scenario. If a logically plausible scenario admits a probability that is strictly > 0, then it is actually *plausible* also in probabilistic terms. Contrariwise, a logically plausible scenario that gets assigned a probability that is forcefully 0 is actually *implausible*. This witnesses in fact that, due to the probabilities attached to the various constraints in M, the fraction of traces belonging to it must be 0.

Example 6. Consider the ProbDeclare model in Figure 1. Its system of inequalities is so that $x_{000} = x_{010} = x_{100} = x_{111} = 0$, since the corresponding constraint scenarios are logically implausible. For the logically plausible scenarios, we instead get the following equalities, once the variables above are removed (being them all equal to 0):

$$\begin{aligned} x_{001} + x_{011} + x_{101} + x_{110} &= 1 \\ x_{101} + x_{110} &= 0.8 \\ x_{011} + x_{110} &= 0.3 \\ x_{001} + x_{011} + x_{101} &= 0.9 \end{aligned}$$

It is easy to see that this system of equations admits only one solution: $x_{001} = 0$, $x_{011} = 0.2$, $x_{101} = 0.7$, $x_{110} = 0.1$. This solution witnesses that scenario S_{001} is implausible, and that the most plausible scenario, holding in 70% of cases, is actually S_{101}, namely the one where after the order is closed, it is eventually accepted, and not refused. In addition, the solution tells us that there are other two outlier scenarios: the first, holding in 20% of cases, is the one where, after the order is closed, it is eventually refused (and not accepted); the second, holding in 10% of cases, is the one where a closed order is accepted and refused. ◁

In general, the system $\mathcal{L}_M$ of inequalities for a ProbDeclare model M may have more than one solution. In this case, we can attach to each constraint scenario a probability interval, whose extreme values are calculated by minimizing and maximizing its corresponding variable over $\mathcal{L}_M$. Since the so-obtained intervals have been computed by analyzing each variable in isolation, not all the combinations of values residing in such intervals are actually consistent. Still, for sure these intervals contain probability values that are overall consistent, and, in addition, they provide a good indicator of which are the most (and less) plausible scenarios. We illustrate this in the next example.

Example 7. Consider the ProbDeclare model in Figure 2. It comes with 4 constraint scenarios, obtained by considering the two constraint formulae `precedence`(sign,close)=¬close$\mathcal{W}$sign and `response`(close,sign)=□(close → ○◇sign), as well as their respective negated formulae ¬sign $\mathcal{U}$ close and ◇(close ∧ ¬○◇sign). All such scenarios are logically plausible, and hence the corresponding

system of inequalities is:

$$\begin{aligned} x_{00} \geq 0 \quad x_{01} \geq 0 \quad x_{10} \geq 0 \quad x_{11} \geq 0 \\ x_{00} + x_{01} + x_{10} + x_{11} = 1 \\ x_{10} + x_{11} = 0.8 \\ x_{01} \quad + x_{11} = 0.1 \end{aligned}$$

This system is very similar to that of Example 5, but here all scenarios are logically plausible, and so it admits multiple solutions. In fact, by calculating the minimum and maximum values for the 4 variables, we get that:

- scenario S_{00}, where the order is closed but consent is not signed, comes with probability interval $[0, 0.1]$;
- scenario S_{01}, where the order is closed and consent is signed afterwards, comes with probability interval $[0, 0.1]$;
- scenario S_{10}, where the order is closed after having signed consent, comes with probability interval $[0.7, 0.8]$;
- scenario S_{11}, where the order is closed and consent is signed at least twice (once before, and once afterwards), comes with probability interval $[0.1, 0.2]$.

This shows that the actual plausibility of each constraint scenario depends on which specific values are selected for the other ones. ◁

4 Discovering ProbDeclare Models from Event Logs

We now show that ProbDeclare models can be discovered from event data using, off-the-shelf, already existing techniques, with a quite interesting guarantee: that the discovered model is for sure consistent.

We start by introducing the usual definition of an event log (only consisting of punctual activity occurrences, without data).

Definition 4. *A log over a set Σ of activities is a multiset of finite traces over Σ.* ◁

We make use of the standard notation $[\cdot]$ for multisets, and use superscript numbers to identify the multiplicity of an element in the multiset.

A plethora of different algorithms have been devised to discover Declare models from event data [7,10,5,19]. In general terms, the vast majority of these algorithms adopts the following approach to discovery. (1) Candidate constraints are generated by analyzing the activities contained in the log. (2) For each constraint, its so-called *support* is computed as the fraction of traces in the log where the constraint holds.

Definition 5 (Constraint Support). *The support of an LTL$_f$ constraint φ in an event log $\mathcal{L} = [\tau_1, \ldots, \tau_n]$ is*

$$supp_{\mathcal{L}}(\varphi) = \frac{|\mathcal{L}_\varphi|}{|\mathcal{L}|}, \text{ where } \mathcal{L}_\varphi = [\tau \in \mathcal{L} \mid \tau \models \varphi]$$ ◁

(3) Candidate constraints are filtered by retaining only those whose support exceeds a given threshold. (4) Further filters (e.g., considering the "relevance" of a constraint [4]) are applied. (5) The overall model is checked for consistency, operating with different strategies if it is not. This last step is necessary since constraints with high support, but still less than 1, may actually conflict with each other [3].

Notably, thanks to the semantics of PLTL$_f^0$, we can use this approach off-the-shelf to discover ProbDeclare constraints: *we just use the constraint support as its associated*

probability. We can also relax step (3), e.g., to retain constraints with a very low support - this would in fact imply that their negated versions have a very high support.

Example 8. Consider event log $\mathcal{L} = [\langle \mathsf{close}, \mathsf{acc} \rangle^7, \langle \mathsf{close}, \mathsf{ref} \rangle^2, \langle \mathsf{close}, \mathsf{acc}, \mathsf{ref} \rangle^1]$, capturing the evolution of 10 orders, 7 of which have been closed and then accepted, 2 of which have been closed and then refused, and 1 of which has been closed, then accepted, then refused. The support of constraint `response`(close,acc) is $8/10 = 0.8$, witnessing that 8 traces satisfy such a constraint, whereas 2 violate it. This corresponds exactly to the interpretation of probability 0.8 for the probabilistic `response`(close,acc) constraint in Figure 1. Actually, the entire ProbDeclare model of Figure 1 can be discovered from $\mathcal{L}$ by considering the 6 constraints contained in that model and their corresponding support over $\mathcal{L}$. ◁

A second key observation is that once the procedure above is used to discover ProbDeclare constraint, it is not necessary anymore to perform step 5: as shown next, since each probabilistic constraint carries its own support, the overall discovered model is guaranteed to be consistent.

Theorem 1. *Let Σ be a set of activities, and $\mathcal{L}$ be an event log over Σ. Let $\mathcal{C} = \{\langle \varphi_1, p_1 \rangle, \ldots, \langle \varphi_n, p_n \rangle\}$ be a set of probabilistic constraints discovered from $\mathcal{L}$, such that for each $i \in \{1, \ldots, n\}$, we have that $p_i = supp_{\mathcal{L}}(\varphi_i)$. Then, the ProbDeclare model $\langle \Sigma, \mathcal{C} \rangle$ is consistent.* ◁

Proof. Recall that $\langle \Sigma, \mathcal{C} \rangle$ is consistent if its corresponding PLTL$_f^0$ formula $\Phi := \{\circledcirc_{p_1} \varphi_1, \ldots, \circledcirc_{p_n} \varphi_n\}$ is satisfiable. To show this, we simply use $\mathcal{L}$ to build a model of Φ. For every set $I \subseteq \{1, \ldots, n\}$, let φ_I be the LTL$_f$ formula

$$\varphi_I := \bigwedge_{i \in I} \varphi_i \wedge \bigwedge_{i \notin I} \neg \varphi_i,$$

and let $\mathcal{L}_I$ be the sublog of $\mathcal{L}$ containing all the traces that satisfy φ_I. Note that the sublogs $\mathcal{L}_I$ form a partition of $\mathcal{L}$; that is, every trace appears in exactly one such $\mathcal{L}_I$. For each I such that $\mathcal{L}_I$ is not empty, choose a representative $t_I \in \mathcal{L}_I$ and let $p_I := \frac{|\mathcal{L}_I|}{|\mathcal{L}|}$ be the fraction of traces that belong to $\mathcal{L}_I$. We build a PLTL$_f^0$ interpretation $\mathcal{I}$ by setting $\mathcal{I} := \{(t_I, p_I) \mid \mathcal{L}_I \neq \emptyset\}$. It remains to be shown that $\mathcal{I}$ is a model of Φ. Consider a constraint $\langle \varphi, p \rangle \in \mathcal{C}$; we need to show that $p = \sum_{(t_I, p_I) \in \mathcal{I}. t_I \models \varphi} p_I$. Note that by construction, since $\mathcal{L}_I$ form a partition of $\mathcal{L}$, $\sum_{(t_I, p_I) \in \mathcal{I}. t_I \models \varphi} p_I$ is in fact the fraction of traces that satisfy φ. On the other hand, p is also the support of φ; that is, the proportion of traces satisfying φ. Hence, both values are equal, and $\mathcal{I}$ is a model of Φ. ⊣

Thanks to this theorem, we have that probabilistic constraints can be discovered in a *purely local* way, having the guarantee that they will never conflict with each other. Obviously, non-local filters can still prove useful to prune implied constraints and select the most relevant ones. Also, note that the probabilities of the discovered constraints can be easily adjusted when new traces are added to the log, by incrementally recomputing the support values after checking how many new traces satisfy the various constraints.

5 Monitoring Probabilistic Constraints

In Section 3.2, we have shown how we can take a ProbDeclare model and generate its constraint scenarios, together with their corresponding probability intervals. We now

describe how this technique can be directly turned into an operational probabilistic monitoring and conformance checking framework.

Let $M = \langle \Sigma, \mathcal{C} \rangle$ be a ProbDeclare model with n probabilistic constraints. For simplicity of exposition, we do not here distinguish between crisp and genuinely probabilistic constraints, nor prune away implausible scenarios: the produced monitoring results do not change, but obviously our implementation, presented at the end of this section, takes into account these aspects for optimization reasons.

M generates 2^n constraint scenarios. As discussed in Section 3.2, each scenario S comes with a corresponding characteristic LTL_f formula, which amounts to the conjunction of positive and negated constraints in $\mathcal{C}$, where the decision of which ones are taken positive and which negative is defined by the scenario itself. We denote such a formula by $formula(S)$. For example, if $\mathcal{C} = \{\langle \varphi_1, p_1 \rangle, \langle \varphi_2, p_2 \rangle, \langle \varphi_3, p_3 \rangle\}$, then $formula(S_{101}) = \varphi_1 \wedge \neg\varphi_2 \wedge \varphi_3$.

In addition, each scenario S comes with its probability interval, calculated by minimizing and maximizing its probability variable in the system of inequalities $\mathcal{L}_M$. We denote such a probability interval by $prob(S)$. By considering Example 7, we have, e.g., that $prob(S_{10}) = [0.7, 0.8]$.

Since the characteristic formula of a scenario is in standard LTL_f, we can construct a *scenario monitor* by recasting well-known techniques [13,1]. Specifically, given an LTL_f formula φ over a set Σ of activities, and a partial trace τ representing an ongoing process execution, a monitor outputs one of the four following truth values:

- τ *(permanently) satisfies* φ, if φ is currently satisfied ($\tau \models \varphi$), and φ stays satisfied no matter how the execution continues, that is, for every possible continuation trace τ' over Σ, we have $\tau \cdot \tau' \models \varphi$ (the $\cdot$ operator denotes the concatenation of two traces);
- τ *possibly satisfies* φ, if φ is currently satisfied ($\tau \models \varphi$), but φ may become violated in the future, that is, there exists a continuation trace τ' over Σ such that $\tau \cdot \tau' \not\models \varphi$;
- τ *possibly violates* φ, if φ is currently violated ($\tau \not\models \varphi$), but φ may become satisfied in the future, that is, there exists a continuation trace τ' over Σ such that $\tau \cdot \tau' \models \varphi$;
- τ *(permanently) violates* φ, if φ is currently violated ($\tau \not\models \varphi$), and φ stays violated no matter how the execution continues, that is, for every possible continuation trace τ' over Σ, we have $\tau \cdot \tau' \not\models \varphi$.

In [13,1], it is shown that a monitor producing such outputs can be seamlessly obtained by constructing the finite-state automaton $\mathcal{A}_\varphi$ for φ, determinizing such an automaton, and finally assigning each automaton state to one of the four truth values.

We then proceed as follows. For each plausible constraint scenario S over M, we construct the finite-state automaton $\mathcal{A}_{formula(S)}$, and turn it into a monitor as described above.[5] We then track the evolution of a running trace by delivering its events to *all* such monitors in parallel, returning the truth values they produce. Note that, at runtime, we do not know in which scenario the trace will fall when completed. Hence, we actually do not know the exact truth value of the trace. For this reason, we compute the truth value of the trace probabilistically by aggregating the probabilities of the scenarios that produce the same truth value. In particular, we compute the *aggregated probability interval* for each truth value, by taking the system of inequalities $\mathcal{L}_M$ and calculating the extreme values of the aggregated interval by minimizing/maximizing the sum of

[5] Implausible scenarios are irrelevant, since they produce an output that is associated to probability 0, and would be therefore discarded when computing the aggregated probability intervals.

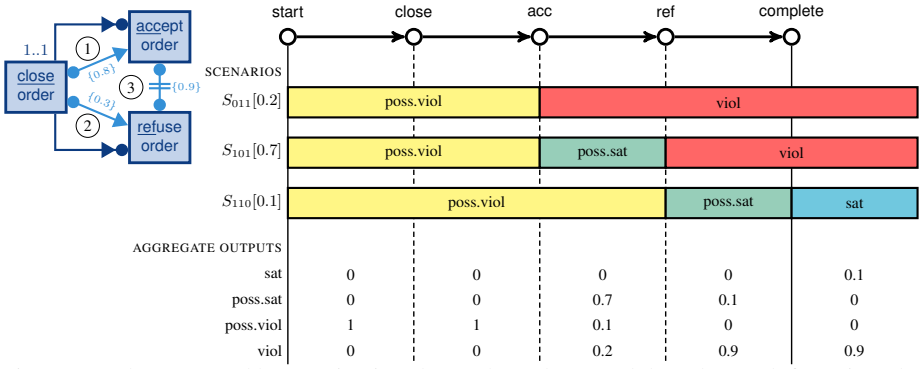


Fig. 3: Result computed by monitoring the ProbDeclare model on the top left against the trace ⟨close, acc, ref⟩, which conforms to the outlier constraint scenario where the two `response` constraints are satisfied, while the `not-coexistence` one is violated.

trace ⟨close, sign⟩ trace ⟨sign, close⟩

Fig. 4: Output of the implemented tool on the example in Fig.2.

the probability variables associated to the scenarios that produce that truth value. The aggregated probabilities give an indication of how probable is that the trace can be associated to each specific truth value.

When a trace finishes (which is signalled by a special, *complete* event) all monitors currently outputting *possible satisfaction* turn to *permanent satisfaction*, and those

outputting *possible violation* turn to *permanent violation* (since no further events will occur to subvert the current situation). Hence, when a trace finishes, it either permanently violates all scenarios (thus being classified as a non-conforming one), or permanently violates all scenarios but one, which is permanently satisfied and consequently witnesses that the trace conforms to that scenario. Notably, this can be instrumental to classify traces into process variants.

Example 9. Consider the ProbDeclare model in Figure 1 with its three plausible scenarios (recall that four scenarios are logically plausible there, but one of those has probability 0, so only three remains to be monitored). Figure 3 shows the result produced when monitoring a trace that at some point appears to belong to the most plausible scenario, but in the end turns out to conform to the less plausible one. From the image, we can also clearly see that the trace consisting only of a close order activity would be judged as non-conforming, as it would violate all scenarios. ◁

The aforementioned probabilistic monitoring technique has been fully implemented.[6] For solving systems of inequalities, we use the Java based LP solver provided at `http://lpsolve.sourceforge.net/5.5/`. The implementation comes with various optimizations. First and foremost, scenarios are computed by directly imposing that crisp constraints with probability 1 must hold in their positive form in all scenarios. Second, only plausible scenarios are retained for monitoring. Third, the results obtained by minimizing and maximizing for aggregate probability variables are cached, so as to avoid solving multiple times the same problem. Fig.4 shows the output of the implemented monitoring tool on the example in Fig.2 and for two different traces.[7] Here, the aggregated probability intervals are shown with a dark gray or light gray background depending on whether their midpoint is closer to 1 or to 0, respectively, thus giving an indication of the most probable truth values for the monitored trace. The first trace (on the left) is classified as belonging to scenario S_{01} and is an outlier because this scenario has low probability (corresponding to a probability interval of $prob(S_{01}) = [0.0, 0.1]$). The second trace (on the right) is classified as belonging to the highly plausible scenario S_{10} (corresponding to a probability interval of $prob(S_{10}) = [0.7, 0.8]$).

6 Conclusion

In this paper, we have introduced the notion of probabilistic business constraint and demonstrated how this notion affects the outcomes of standard process mining approaches based on Declare when standard Declare is replaced by its probabilistic counterpart. We have introduced a framework for monitoring a trace with respect to a set of probabilistic constraints by computing, during the evolution of the trace, the most probable truth values for it. The framework also classifies completed traces as violating a given probabilistic model or as belonging to a certain constraint scenario (i.e., satisfying a certain combination of probabilistic constraints).

For future work, we plan to better investigate the influence of probabilistic constraints on the state-of-the-art techniques for declarative process mining. We also plan

[6] `https://bitbucket.org/fmmaggi/probabilisticmonitor/src/master/`

[7] In the screenshots, 1 and 2 represent the probabilistic constraints labeled with 1 and 2 in Fig.2, whereas 3 represents the crisp constraint in the same example.

to extend the notion of probabilistic process model and probabilistic monitoring to the context of imperative process modeling languages.

References

1. De Giacomo, G., De Masellis, R., Grasso, M., Maggi, F.M., Montali, M.: Monitoring business metaconstraints based on LTL & LDL for finite traces. In: Proc. of BPM. LNCS, vol. 8659. Springer (2014)
2. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proc. of IJCAI. AAAI Press (2013)
3. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Inf. Syst. **64** (2017)
4. Di Ciccio, C., Maggi, F.M., Montali, M., Mendling, J.: On the relevance of a business constraint to an event log. Inf. Syst. **78** (2018)
5. Di Ciccio, C., Mecella, M.: On the discovery of declarative control flows for artful processes. ACM Trans. Management Inf. Syst. **5**(4), 24:1–24:37 (2015)
6. Kovtunova, A., Peñaloza, R.: Cutting diamonds: A temporal logic with probabilistic distributions. In: Proc. of KR. AAAI Press (2018)
7. Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Inducing declarative logic-based models from labeled traces. In: Proc. of BPM. LNCS, vol. 4714. Springer (2007)
8. Leemans, S.J.J., Syring, A.F., van der Aalst, W.M.P.: Earth movers' stochastic conformance checking. In: Proc. of BPM Forum. LNBIP, vol. 360. Springer (2019)
9. Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Inf. Syst. **54** (2015)
10. Maggi, F.M., Chandra Bose, R.P.J., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: Proc. of CAiSE. LNCS (2012)
11. Maggi, F.M., Montali, M., Peñaloza, R.: Probabilistic temporal logic over finite traces (technical report). CoRR **abs/1903.04940** (2019), `http://arxiv.org/abs/1903.04940`
12. Maggi, F.M., Montali, M., Peñaloza, R.: Temporal logics over finite traces with uncertainty. In: Proc. of AAAI. AAAI Press (2020), to appear.
13. Maggi, F.M., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: An approach based on colored automata. In: Proc. of BPM. LNCS, Springer (2011)
14. Maggi, F.M., Westergaard, M., Montali, M., van der Aalst, W.M.P.: Runtime verification of LTL-based declarative process models. In: Proc. of RV. LNCS, vol. 7186. Springer (2011)
15. Montali, M.: Specification and Verification of Declarative Open Interaction Models: a Logic-Based Approach, LNBIP, vol. 56. Springer (2010)
16. Montali, M., Pesic, M., van der Aalst, W.M.P., Chesani, F., Mello, P., Storari, S.: Declarative specification and verification of service choreographies. TWEB **4**(1) (2010)
17. Ognjanovic, Z.: Discrete linear-time probabilistic logics: Completeness, decidability and complexity. J. Log. Comput. **16**(2) (2006)
18. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. In: Proc. of EDOC. IEEE Computer Society (2007)
19. Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and customisable declarative process mining with SQL. In: CAiSE 2016. pp. 290–305 (2016)